
Assignment 1

Name: Chirag Sarda

Roll No.: 20110047

The code are pushed on the [github](#)

Coding Question

I have used IPv4 and TCP protocol for the connection \

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind(addr_tuple)
# AF_INET --- address family of IPv4
# SOCK_STREAM --- connection oriented tcp protocol
```

User can run the server_code.py first, it will give the IP address and port number. After we can run client_code.py, and then specify the IP address and port number of the server.

After that client code will ask the user to specify the crypto mode. One can give 1,2 and 3 corresponding to modes given in the question. Crypto layer is attacked at the zeroth index of the sending string, and it is not encrypted and decrypted. Server and Client can get to know which crypto mode is used for by reading the zeroth index.

```
PS E:\5thSem\CS433\Assignment\Assignment1\CLIENT> python .\client_code.py
[RUNNING] Client started running
IP address of the host - 192.168.56.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which cypto mode you want to use 2
User command -
```

```
PS E:\5thSem\CS433\Assignment\Assignment1\SERVER> python .\server_code.py
[RUNNING] Server running at IP = 192.168.56.1 and port number = 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('192.168.56.1', 56120)

```

After that it will ask for the user to give command:

1. CWD command: Client is sending the encrypted message (according to the crypto mode chosen earlier)

Client:

```
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 2EYF
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER
User command -
```

Server:

```
The packet received at the server is 2EYF
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
█
```

2. LS command:

Client:

```
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 2NU
[RESPONSE] server response: a.exe
Home
Library
server_code.py
User
User command - █
```

Server:

```
The packet received at the server is 2NU
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
█
```

3. CD Command

Client:

```
User command - CD User
[SENDING] CD commnad to Server
Sending packet to the server 2EF Wugt
[RESPONSE] server response: [OK]
User command - █
```

Server:

```
The packet received at the server is 2EF Wugt
[RECEIVED] CD commnad at Server
[SENT] Server sent response to client
█
```

Checking whether our CD command is successful or not

```
User command - CD User
[SENDING] CD commnad to Server
Sending packet to the server 2EF Wugt
[RESPONSE] server response: [OK]
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 2EYF
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER\User
User command - █
```

4. UPD Command

Client:

```

User command - UPD gt.txt
[SENDING] UPD commnad to Server
[NOK] File not found at the client
User command - UPD yt.txt
[SENDING] UPD commnad to Server
Sending packet to the server 2WRF av.vzv vjku ku vjg av vgzv hkng^&&*7
[RESPONSE] server response: [OK]
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 2NU
[RESPONSE] server response: ht.txt
yt.txt
User command - []

```

Server:

```

The packet received at the server is 2EYF
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2WRF av.vzv vjku ku vjg av vgzv hkng^&&*7
[RECEIVED] UPD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2NU
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
[]

```

We can check whether the file is being uploaded or not by checking list of all the file (LS command) if file is not found it will display error ([NOK] file not found at the client)

As we can see that it returns yt.txt in the last line which means our upload was correct

5. DWD command

Client:

```

User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 2NU
[RESPONSE] server response: ht.txt
yt.txt
User command - DWD htt.txt
[SENDING] DWD commnad to Server
Sending packet to the server 2FYF jvv.vzv
File Not Found on the server
User command - DWD ht.txt
[SENDING] DWD commnad to Server
Sending packet to the server 2FYF jv.vzv
ht.txt downloaded successfully
User command - []

```

Server:

```

[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2FYF jvv.vzv
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2FYF jv.vzv
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client

```

All the downloaded file saves in the Current working directory of client, if the file is not found on the server then it will display file not found on the server

6. `exit()`: to close the connection between the server and client

Client:

```
Which cypto mode you want to use 1
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 1LS
[RESPONSE] server response: a.exe
Home
Library
server_code.py
User
User command - ls
[SENDING] ls commnad to Server
Sending packet to the server 1ls
[RESPONSE] server response: Command Not Found
User command - xds
[SENDING] xds commnad to Server
Sending packet to the server 1xds
[RESPONSE] server response: Command Not Found
User command - █
```

Server:

```
The packet received at the server is 1LS
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1ls
[RECEIVED] ls commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1xds
[RECEIVED] xds commnad at Server
[SENT] Server sent response to client
█
```

7. else situation: If some random command is typed: Code will display message (Command not found), but connection will not get close.

Client

```
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 1CWD
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER
User command - fldfld
[SENDING] fldfld commnad to Server
Sending packet to the server 1fldfld
[RESPONSE] server response: Command Not Found
User command - █
```

Server:

```
The packet received at the server is 1CWD
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1fldfld
[RECEIVED] fldfld commnad at Server
[SENT] Server sent response to client
█
```

Sending DWD in all the 3 different modes

1: Plain Text

Client:

```
PS E:\5thSem\CS433\Assignment\Assignment1\CLIENT> python .\client_code.py
[RUNNING] Client started running
IP address of the host - 192.168.56.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which cypto mode you want to use 1
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 1CWD
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER
User command - CD Home
[SENDING] CD commnad to Server
Sending packet to the server 1CD Home
[RESPONSE] server response: [OK]
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 1LS
[RESPONSE] server response: a.cpp
C1.txt
C2.txt
C3.txt
gt.txt
User command - DWD C1.txt
[SENDING] DWD commnad to Server
Sending packet to the server 1DWD C1.txt
C1.txt downloaded successfully
User command - []
```

Server:

```
PS E:\5thSem\CS433\Assignment\Assignment1\SERVER> python .\server_code.py
[RUNNING] Server running at IP = 192.168.56.1 and port number = 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('192.168.56.1', 56549)
The packet received at the server is 1CWD
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1CD Home
[RECEIVED] CD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1LS
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
The packet received at the server is 1DWD C1.txt
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
[]
```

2. Substitution Mode:

Client:

```

[RECEIVED] client connection closed
PS E:\5thSem\CS433\Assignment\Assignment1\CLIENT> python .\client_code.py
[RUNNING] Client started running
IP address of the host - 192.168.56.1
Port no of the host - 5050
[CREATED] clinet socket successfully created
[REQUESTING] client requesting for connection
Which cypto mode you want to use 2
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 2EYF
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER
User command - CD Home
[SENDING] CD commnad to Server
Sending packet to the server 2EF Jqog
[RESPONSE] server response: [OK]
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 2NU
[RESPONSE] server response: a.cpp
C1.txt
C2.txt
C3.txt
gt.txt
User command - DWD C2.txt
[SENDING] DWD commnad to Server
Sending packet to the server 2FYF E4.vzv
C2.txt downloaded successfully
User command - 

```

Server:

```

PS E:\5thSem\CS433\Assignment\Assignment1\SERVER> python .\server_code.py
[RUNNING] Server running at IP = 192.168.56.1 and port number = 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('192.168.56.1', 56559)
The packet received at the server is 2EYF
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2EF Jqog
[RECEIVED] CD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2NU
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
The packet received at the server is 2FYF E4.vzv
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client

```

3 Transpose Mode:

Client:

```

PS E:\5thSem\CS433\Assignment\Assignment1\CLIENT> python .\client_code.py
[RUNNING] Client started running
IP address of the host - 192.168.56.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which cypto mode you want to use 3
User command - CWD
[SENDING] CWD commnad to Server
Sending packet to the server 3DWC
[RESPONSE] server response: E:\5thSem\CS433\Assignment\Assignment1\SERVER
User command - CD Home
[SENDING] CD commnad to Server
Sending packet to the server 3DC emoH
[RESPONSE] server response: [OK]
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 3SL
[RESPONSE] server response: a.cpp C1.txt C2.txt C3.txt gt.txt
User command - DWD C3.txt
[SENDING] DWD commnad to Server
Sending packet to the server 3DWD txt.3C
C3.txt downloaded successfully
User command - exit()
[SENDING] exit() commnad to Server
Sending packet to the server 3)(tixe
[RESPONSE] server response: exit()
[CLOSED] Clinet connection closed
PS E:\5thSem\CS433\Assignment\Assignment1\CLIENT>

```

Server:

```

PS E:\5thSem\CS433\Assignment\Assignment1\SERVER> python .\server_code.py
[RUNNING] Server running at IP = 192.168.56.1 and port number = 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('192.168.56.1', 56578)
The packet received at the server is 3DWC
[RECEIVED] CWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 3DC emoH
[RECEIVED] CD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 3SL
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
The packet received at the server is 3DWD txt.3C
[RECEIVED] DWD commnad at Server
[SENT] Server sent response to client
The packet received at the server is 3)(tixe
[RECEIVED] exit() commnad at Server
[SENT] Server sent response to client
[CLOSED] server connection closed
PS E:\5thSem\CS433\Assignment\Assignment1\SERVER>

```

Challenges

1. Once we do the CD command then after that we do CWD it was giving the original directory not the that was changed by the previous CD command. For this have declared a `current_working_directory` variable before the connection is formed.
2. Transporting the file across the socket (UPD and DWD)
3. Applying the crypto Layer (How to decide when to apply which crypto layer). How server will know which crypto Layer we have to use. For this I just added the mode at the 0th index so that server and client will no which crypto mode we are using

4. Making the connection persistent. For that I used the while loop. Received
5. Sending the file is easy as the data is in string format one can directly apply the crypto modes but when it comes to image and video file there, we have to transport information in bytes (in binary), it becomes very complex to handle the crypto mode. My code does not handle this

Checking for the correctness of the encryption layer by wireshark:

For this I have run the code on the kali as I have wireshark on the kali in Virtual Box:

Using the LS command for the analysis:

Crypto Mode 1:

Client:

```
(kali@kali)~/CS433/Assignment1/CLIENT
$ python client_code.py
[RUNNING] Client started running
IP address of the host - 127.0.1.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which crypto mode you want to use 1
User command - LS
[SENDING] LS command to Server
Sending packet to the server 1LS
[RESPONSE] server response: yt.txt Library User server_code.py Home
User command - []
```

Server:

```
(kali@kali)~/CS433/Assignment1/SERVER
$ python server_code.py
[RUNNING] Server running at IP - 127.0.1.1 and port number - 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('127.0.0.1', 53880)
The packet received at the server is 1LS
[RECEIVED] LS command at Server
[SENT] Server sent response to client
[]
```

Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.1	TCP	74	53880 → 5050 [SYN]
2	0.000000248	127.0.1.1	127.0.0.1	TCP	74	5050 → 53880 [SYN]
3	0.000012119	127.0.0.1	127.0.1.1	TCP	66	53880 → 5050 [ACK]
4	6.881839800	127.0.0.1	127.0.1.1	TCP	69	53880 → 5050 [PSH]
5	6.881854365	127.0.1.1	127.0.0.1	TCP	66	5050 → 53880 [ACK]
6	6.882027850	127.0.1.1	127.0.0.1	TCP	106	5050 → 53880 [PSH]
7	6.882033553	127.0.0.1	127.0.1.1	TCP	66	53880 → 5050 [ACK]

Frame 4: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface lo, id 0

- Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)
- Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.1
- Transmission Control Protocol, Src Port: 53880, Dst Port: 5050, Seq: 1, Ack: 1, Len: 3
- Data (3 bytes)

```

0000  00 00 00 00 00 00 00 00 00 00 00 00 45 00  .....E
0010  00 37 b3 75 40 00 00 00 00 49 7f 00 00 01 7f 00  7.u@.I
0020  01 01 d2 78 13 ba 5f d4 73 96 de 7e 5b a4 80 18  ~s~[
0030  02 00 ff 2b 00 00 01 01 00 0a de 52 3e 1c 5a 8d  +...R>Z
0040  95 ff 31 4c 53                                     1LS

```


In wireshark and in my code both are showing 1LS, 1 is for crypto mode

Crypto mode = 2

Client:

```
(kali@kali)-[~/CS433/Assignment1/CLIENT]
$ python client_code.py
[RUNNING] Client started running
IP address of the host - 127.0.1.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which crypto mode you want to use 2
User command - LS
[SENDING] LS command to Server
Sending packet to the server 2NU
[RESPONSE] server response: yt.txt Library User server_code.py Home
User command - []
```

Server:

```
(kali@kali)-[~/CS433/Assignment1/SERVER]
$ python server_code.py
[RUNNING] Server running at IP = 127.0.1.1 and port number = 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('
127.0.0.1', 59022)
The packet received at the server is 2NU
[RECEIVED] LS command at Server
[SENT] Server sent response to client
```

Wireshark:

Wireshark packet capture showing a TCP connection and data exchange between 127.0.0.1 and 127.0.1.1. The packet list shows a SYN, ACK, PSH, ACK sequence. The packet details pane shows the data field containing '2NU'.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.1	TCP	74	59022 → 5050 [SYN] Seq=0 Win=65495 Len=0 MSS=654
2	0.000000646	127.0.1.1	127.0.0.1	TCP	74	5050 → 59022 [SYN, ACK] Seq=0 Ack=1 Win=65483 Le
3	0.000012614	127.0.0.1	127.0.1.1	TCP	66	59022 → 5050 [ACK] Seq=1 Ack=1 Win=65536 Len=0
4	6.498091429	127.0.0.1	127.0.1.1	TCP	69	59022 → 5050 [PSH, ACK] Seq=1 Ack=1 Win=65536 Le
5	6.498101174	127.0.1.1	127.0.0.1	TCP	66	5050 → 59022 [ACK] Seq=1 Ack=4 Win=65536 Len=0
6	6.498215465	127.0.1.1	127.0.0.1	TCP	106	5050 → 59022 [PSH, ACK] Seq=1 Ack=4 Win=65536 Le
7	6.498222187	127.0.0.1	127.0.1.1	TCP	66	59022 → 5050 [ACK] Seq=4 Ack=41 Win=65536 Len=0

Frame 4: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface lo, id 0

Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.1

Transmission Control Protocol, Src Port: 59022, Dst Port: 5050, Seq: 1, Ack: 1, Len: 3

Data (3 bytes)

0000 00 00 00 00 00 00 00 00 00 00 00 00 45 00E
0010 00 37 d9 7b 40 00 40 06 62 43 7f 00 00 01 7f 00 7 (@_@_bc
0020 01 01 e6 8e 13 ba 30 7a 67 f9 1a 1f 3c 0b 80 180z g
0030 02 00 ff 2b 00 00 01 01 00 0a de 53 4a a4 5a 0eSJ Z
0040 a4 05 32 4e 55

When LS is changed to NU in by Crypto mode of substitution. Wireshark and our code gives the same answer

Crypto Mode = 3

Client:

```
(kali@kali)-[~/CS433/Assignment1/CLIENT]
$ python client_code.py
[RUNNING] Client started running
IP address of the host - 127.0.1.1
Port no of the host - 5050
[CREATED] client socket successfully created
[REQUESTING] client requesting for connection
Which cypto mode you want to use 3
User command - LS
[SENDING] LS commnad to Server
Sending packet to the server 3SL
[RESPONSE] server response: yt.txt Library User server_code.py Home
User command - []
```

Server:

```
(kali@kali)-[~/CS433/Assignment1/SERVER]
$ python server_code.py
[RUNNING] Server running at IP - 127.0.1.1 and port number - 5050
[LISTENING] server is listening...
[CONNECTION SUCCESSFUL] Server accepts the connection of client at ('
127.0.0.1', 54894)
The packet received at the server is 3SL
[RECEIVED] LS commnad at Server
[SENT] Server sent response to client
```

Wireshark:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	127.0.0.1	127.0.1.1	TCP	74	54894 → 5050 [SYN] Seq=0 Win=0 Len=0
2	0.000000245	127.0.1.1	127.0.0.1	TCP	74	5050 → 54894 [SYN, ACK] Seq=0 Win=0 Len=0
3	0.000001947	127.0.0.1	127.0.1.1	TCP	66	54894 → 5050 [ACK] Seq=1 Win=0 Len=0
4	5.067741789	127.0.0.1	127.0.1.1	TCP	69	54894 → 5050 [PSH, ACK] Seq=1 Win=0 Len=3
5	5.067751424	127.0.1.1	127.0.0.1	TCP	66	5050 → 54894 [ACK] Seq=1 Win=0 Len=0
6	5.067921978	127.0.1.1	127.0.0.1	TCP	106	5050 → 54894 [PSH, ACK] Seq=1 Win=0 Len=3
7	5.067928762	127.0.0.1	127.0.1.1	TCP	66	54894 → 5050 [ACK] Seq=4 Win=0 Len=0

Frame 4: 69 bytes on wire (552 bits), 69 bytes captured (552 bits) on interface lo, id 0

Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)

Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.1

Transmission Control Protocol, Src Port: 54894, Dst Port: 5050, Seq: 1, Ack: 1, Len: 3

Data (3 bytes)

0000 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 45 00 7f 00 00 01 7f 00
0010 00 37 25 c7 40 00 40 06 15 d8 7f 00 00 01 7f 00 7f 00 00 00 00 00 00
0020 01 01 d6 6e 13 ba c7 a7 22 c4 88 80 83 a0 80 18 00 00 00 00 00 00
0030 02 00 ff 2b 00 00 01 01 00 0a de 54 18 a7 5a 0f 00 00 00 00 00 00 00
0040 77 9f 33 53 4c 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3SL

When LS is changed to SL in by Crypto mode of substitution. Wireshark and our code gives the same answer

For other commands we can similarly do this.

Numerical Question

Name: Chirag Sarda

Roll no: 20110047

Question

→ ~~Message~~ Message ~~Size~~ Size = 100 KB
 Meta data = 100 B / packet

Given → propagation delay = 0 s
 processing delay = 0 s

(9) 1 packet

→ No need to worry for other packet as there is only one packet

$$\begin{aligned} \text{packet size} &= \text{Message size} + 100 \text{ B} \\ &= 100 \text{ KB} + 100 \text{ B} \\ &= (100 \times 1000 + 100) \text{ B} \\ &= (100100) \text{ B} \\ &= 100.1 \text{ KB} \end{aligned}$$

Here T_{trans} Transmission delay

$$\text{Total Time to Reach B} = T_{\text{Link 1}} + T_{\text{Link 2}} + T_{\text{Link 3}}$$

$$T_{\text{delay}} = \frac{\text{Packet Size}}{\text{Bandwidth}}$$

$$T_{\text{Link 1}} = \frac{100.1 \text{ KB}}{400 \text{ Mb/s}} = \frac{1.001 \times 10^5 \times 8}{4 \times 10^8} \text{ s} = 2.002 \text{ ms}$$

$$T_{\text{Link 2}} = \frac{100.1 \text{ KB}}{100 \text{ Mb/s}} = 4(2.002 \text{ ms}) = 8.008 \text{ ms}$$

$$T_{\text{Link 3}} = \frac{100.1 \text{ KB}}{200 \text{ Mb/s}} = 2(2.002) = 4.004 \text{ ms}$$

$$\text{Total Time} = 2.002 + 8.008 + 4.004 \\ = 14.014 \text{ ms}$$

⑥ → 10 packets

$$\begin{aligned} \rightarrow \text{Total packet size} &= 100 \text{ B} + \frac{10 \text{ KB}}{10} \\ &= 100 \text{ B} + 10000 \\ &= 10.1 \text{ KB} \end{aligned}$$

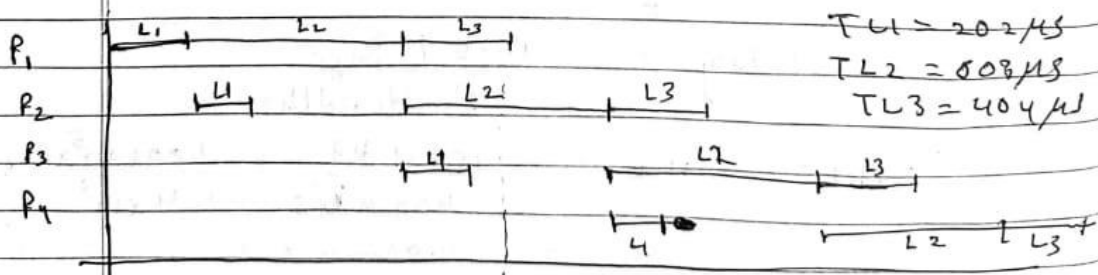
→ Calculating the Transmitter delay for individual link.

$$\begin{aligned} \text{for Link 1} &= \frac{10.1 \text{ KB}}{400 \text{ Mb/s}} = \frac{10.1 \times 10^3 \times 8}{400 \times 10^6} = 0.202 \text{ ms} \\ &= 202 \mu\text{s} \end{aligned}$$

$$\text{for Link 2} = \frac{10.1 \text{ KB}}{120 \text{ Mb/s}} = 4(202 \mu\text{s}) = 808 \mu\text{s}$$

$$\text{for Link 3} = \frac{10.1 \text{ KB}}{200 \text{ Mb/s}} = 2(202 \mu\text{s}) = 404 \mu\text{s}$$

For total Time lets look at way individual packets arrive



if 4 packets then $TL1 + 4 \times TL2 + TL3$

- Although the diagram looks confusing as it suggest the Propagation delay but it is not
- it is showing Grant Chart
- P1 leaves A at $202 \mu s$, leaves B at $808 \mu s$
leaves C at $404 \mu s$ (It needs not to worry about others.)

Packet 2

- P2 can ~~enter~~ ^{enter} A after P1 has left but it can't ~~for~~ ~~leave~~ ~~leave~~ A until P1 has left R1

- This causes the P1 to ^{wait} ~~leave~~ ~~wait~~ which increase our time

→ For 10 packets

$$\begin{aligned} \text{Total Time} &= 202 \mu s + 10 (806 \mu s) + 404 \mu s \\ &= (8080 + 202 + 404) \mu s \\ &= 8686 \mu s \\ &= 8.686 \text{ ms} \end{aligned}$$

① 50 packets

$$\text{Total packet size} = 100 \text{ B} + \frac{100 \text{ KB}}{50} = 2.1 \text{ KB}$$

Transmission delay

$$\text{Link 1} = \frac{2.1 \text{ KB}}{400 \text{ Mbps}} = \frac{2.1 \times 10^3 \times 8}{400 \times 10^6} = 42 \mu s$$

$$\text{Link 2} = 4 (\text{Link 1}) = 168 \mu s$$

$$\text{Link 3} = 2 (\text{Link 1}) = 84 \mu s$$

$$\text{Total Time} = (42 + 84 \times 168 + 84) \mu s = 14142 \mu s = 14.142 \text{ ms}$$

$$= 8.526 \text{ ms}$$

any packet count = 50

(d) packet Count = 100
 Packet Size = $100 \text{ B} + \frac{100 \text{ kB}}{100} = 1.1 \text{ kB}$

Transmission delay for

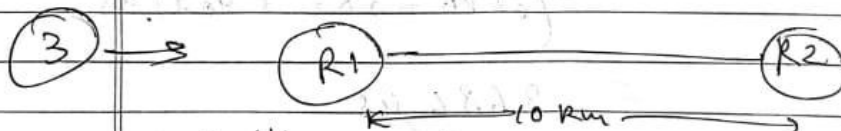
Link 1 = $\frac{1.1 \text{ kB}}{400 \text{ Mb/s}} = \frac{1.1 \times 10^3 \times 8}{400 \times 10^6} = 22 \mu\text{s}$

Link 2 = $4 (\text{Link 1}) = 88 \mu\text{s}$

Link 3 = $2 (\text{Link 1}) = 44 \mu\text{s}$

Total Time $22 + 100(88) + 44$
 $22 + 8800 + 44$
 $8866 \mu\text{s}$
 8.866 ms

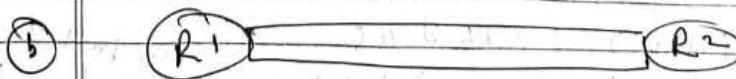
Clearly out all 8.526 ms is the lowest delivery time.



Bandwidth = 100 Gbps

$v = \frac{2}{3} c = \frac{2}{3} \times 3 \times 10^8 = 2 \times 10^8 \text{ m/s}$

(a) → Propagation delay Delay in Medium while transferring
 $= \frac{\text{Length}}{\text{velocity}} = \frac{5 \times 10^3}{2 \times 10^8} = 50 \mu\text{s}$



Maximum bit in the medium (Link)

$$= \text{Propagation time} \times \text{Bandwidth}$$

$$= 100 \times 10^9 \times 50 \times 10^{-6} = 5 \times 10^6 \text{ b}$$

$$= 5 \text{ Mb}$$

(C) ~~area~~

$$\text{Bit Width} = \frac{\text{Speed}}{\text{Bandwidth}} = \frac{2 \times 10^8}{10^{11}} = 2 \text{ mm}$$

(4) \rightarrow RTT (Round Trip Time) = 10 ms

web page size \rightarrow 1 kb

object size \rightarrow 100 kb

10 object Load

(5) HTTP (1.0) NonPersistent Connection

$$\rightarrow \sum_{i=1}^n (2 \text{ RTT} + \text{file transmission time})$$

n = total object loads

2 RTT \rightarrow 1 initial TCP/UDP connection (since http so mainly TCP)
2 \rightarrow http request/ response

~~or 10 (2 RTT + file transmission time)~~

Individual
11 (2 RTT + file transmission time)

\rightarrow 1 for web page

\rightarrow 10 for its object

$$= 22 \text{ RTT} + t \quad (\text{Assuming total transmission time be } t)$$

$$= 220 \text{ ms} + t$$

it will include the time for web page & 10 object

$$t = t_1 + 10 t_2$$

for object

⑥ Persistent Connection (HTTP 1.1)

$$\text{Total Time} = \underbrace{RTT}_{\text{Initial Connection}} + \sum_{i=1}^m \underbrace{(RTT + \text{file transfer})}_{\text{http connection}}$$

$$= (1+11) RTT + t \quad \text{All file transfer time}$$

$$= 12 RTT + t$$

$$= 120 \text{ ms} + t$$

$t = t_1 + 10 t_2$
 $\downarrow \quad \quad \downarrow$
 for web page for object loaded by web page

⑦ HTTP 1.2.0 (persistent + pipelined) → data frames of 1 KB

$$\text{Total Time} = \underbrace{RTT}_{\text{Initial TCP}} + \underbrace{RTT}_{\text{http for web page}} + \underbrace{RTT}_{\text{http for object}} + \text{file transfer time}$$

$$30 \text{ ms} + t$$

→ t_1 would depend on size of web page similarly t_2 would depend on size of object

If assumed $t_1 \propto \text{Size}$

For Non persistent = Total Time = $220 + (1+1000) t_1'$

persistent = $120 + (1+1000) t_1'$

pipelined + persistent = $30 \text{ ms} + 1001 t_1'$

New

1. ARP Address Resolution Protocol [RFC 826](#) is used to convert the IP address into physical address

34	10.833758847	216.58.203.36	10.0.2.15	UDP	77	443 → 48079	Len=33
35	10.854116140	10.0.2.15	216.58.203.36	UDP	77	48079 → 443	Len=33
36	10.903023969	216.58.203.36	10.0.2.15	UDP	116	443 → 48079	Len=72
37	10.903178992	216.58.203.36	10.0.2.15	UDP	73	443 → 48079	Len=29
38	10.903420692	10.0.2.15	216.58.203.36	UDP	83	48079 → 443	Len=39
39	10.918486816	216.58.203.36	10.0.2.15	UDP	73	443 → 48079	Len=29
40	11.775509677	10.0.2.15	172.217.174.227	TCP	56	[TCP Dup ACK 5#1] 56192 → 80 [ACK] Seq=1 Ack=1 Win=63882 Len=	
41	11.775545808	10.0.2.15	172.217.174.227	TCP	56	[TCP Dup ACK 6#1] 56204 → 80 [ACK] Seq=1 Ack=1 Win=63882 Len=	
42	11.776648868	172.217.174.227	10.0.2.15	TCP	62	[TCP Dup ACK 7#1] [TCP ACKed unseen segment] 80 → 56192 [ACK]	
43	11.776646960	172.217.174.227	10.0.2.15	TCP	62	[TCP Dup ACK 8#1] [TCP ACKed unseen segment] 80 → 56204 [ACK]	
44	12.285204809	10.0.2.15	117.18.237.29	TCP	56	[TCP Dup ACK 9#1] 37032 → 80 [ACK] Seq=1 Ack=1 Win=63920 Len=	
45	12.287046244	117.18.237.29	10.0.2.15	TCP	62	[TCP Dup ACK 10#1] [TCP ACKed unseen segment] 80 → 37032 [ACK]	
46	12.798117282	10.0.2.15	172.217.174.227	TCP	56	[TCP Dup ACK 11#1] 56188 → 80 [ACK] Seq=1 Ack=1 Win=63882 Len=	
47	12.798351227	172.217.174.227	10.0.2.15	TCP	62	[TCP Dup ACK 12#1] [TCP ACKed unseen segment] 80 → 56188 [ACK]	
48	15.407430548	PcsCompu	22:46:4f	ARP	44	Who has 10.0.2.2? Tell 10.0.2.15	
49	15.407531297	10.0.2.15	172.217.174.227	TCP	56	[TCP Dup ACK 13#1] 36238 → 80 [ACK] Seq=1 Ack=1 Win=63882 Len=	
50	15.407722916	RealtekU	12:35:02	ARP	62	10.0.2.2 is at 52:54:00:12:35:02	
51	15.407723093	172.217.174.227	10.0.2.15	TCP	62	[TCP Dup ACK 14#1] [TCP ACKed unseen segment] 80 → 36238 [ACK]	
Frame 48: 44 bytes on wire (352 bits), 44 bytes captured (352 bits) on interface any, id 0							
Linux cooked capture v1							
Address Resolution Protocol (request)							

2. QUIC: RCF 8999 to 9002, new reliable and secure transport layer protocol, made to replace the TCP

24	2.967985423	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=1 Ack=668 Win=65535 Len=0
25	2.967985536	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=1 Ack=1481 Win=65535 Len=0
26	2.975438835	172.217.167.164	10.0.2.15	QUIC	1481	Protected Payload (KP0), DCID=d6a0d7
27	2.975576195	172.217.167.164	10.0.2.15	QUIC	659	Protected Payload (KP0), DCID=d6a0d7
28	2.975612755	172.217.167.164	10.0.2.15	QUIC	72	Protected Payload (KP0), DCID=d6a0d7
29	2.976583336	10.0.2.15	172.217.167.164	QUIC	201	Protected Payload (KP0), DCID=d9483f9c53b4484c
30	2.976653279	10.0.2.15	172.217.167.164	QUIC	115	Protected Payload (KP0), DCID=d9483f9c53b4484c
31	3.084258782	172.217.167.164	10.0.2.15	TLSv1.3	886	Server Hello, Change Cipher Spec, Application Data, Application Data, A
32	3.084274468	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1481 Ack=831 Win=63910 Len=0
33	3.084631703	10.0.2.15	172.217.167.164	TLSv1.3	148	Application Data, Application Data
34	3.084844703	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=831 Ack=1565 Win=65535 Len=0
35	3.085315257	172.217.167.164	10.0.2.15	TLSv1.3	87	Application Data
36	3.085322053	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1565 Ack=862 Win=63910 Len=0
37	3.086272357	10.0.2.15	172.217.167.164	TLSv1.3	87	Application Data
38	3.086443193	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=862 Ack=1596 Win=65535 Len=0
39	3.179557612	172.217.167.164	10.0.2.15	TLSv1.3	2165	Application Data, Application Data
Frame 39: 115 bytes on wire (920 bits), 115 bytes captured (920 bits) on interface any, id 0						
Linux cooked capture v1						
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 172.217.167.164						

3. ICMP: Internet Control Message Protocol ([RFC 792](#)), used for host to host datagram services in the interconnected network

40	3.17959072	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=2574 Win=62780 Len=0
41	3.181892698	172.217.167.164	10.0.2.15	TLSv1.3	87	Application Data
42	3.181119208	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=3002 Win=62780 Len=0
43	3.182180658	172.217.167.164	10.0.2.15	TLSv1.3	95	Application Data
44	3.182199896	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=3041 Win=62780 Len=0
45	3.184943938	10.0.2.15	172.217.167.164	TLSv1.3	95	Application Data
46	3.185429058	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=3041 Ack=1635 Win=65535 Len=0
47	24.097013389	fe80::b4e4:c843:92a	ff02::2	ICMPv6	64	Router Solicitation
48	28.002286595	10.0.2.15	35.244.181.201	TLSv1.2	102	Application Data
49	28.002325848	10.0.2.15	18.161.111.99	TLSv1.2	102	Application Data
50	28.002525942	35.244.181.201	10.0.2.15	TCP	62	443 → 46126 [ACK] Seq=1 Ack=47 Win=65535 Len=0
51	28.002526104	18.161.111.99	10.0.2.15	TCP	62	443 → 43116 [ACK] Seq=1 Ack=47 Win=65535 Len=0
52	28.013224708	35.244.181.201	10.0.2.15	TLSv1.2	102	Application Data
53	28.013235314	10.0.2.15	35.244.181.201	TCP	56	46126 → 443 [ACK] Seq=47 Ack=47 Win=64015 Len=0
54	28.113273500	18.161.111.99	10.0.2.15	TLSv1.2	102	Application Data
55	28.113291083	10.0.2.15	18.161.111.99	TCP	56	43116 → 443 [ACK] Seq=47 Ack=47 Win=62780 Len=0

Frame 47: 64 bytes on wire (512 bits), 64 bytes captured (512 bits) on interface any, id 0
Linux cooked capture v1
Internet Protocol Version 6, Src: fe80::b4e4:c843:92af:7956, Dst: ff02::2
Internet Control Message Protocol v6

4. TLSv1.v2: [RFC 5426](#): Transport layer security protocol. Provides communication security on the internet

58	17.119753422	142.250.192.78	10.0.2.15	TLSv1.2	95 Application Data
59	17.163953164	10.0.2.15	142.250.192.78	TCP	56 46892 → 443 [ACK] Seq=40 Ack=40 Win=62780
60	17.659794022	10.0.2.15	13.227.138.97	TCP	56 [TCP Dup ACK 19#1] 45542 → 443 [ACK] Seq=
61	17.600008531	13.227.138.97	10.0.2.15	TCP	62 [TCP Dup ACK 20#1] [TCP ACKed unseen segme
62	18.116208830	10.0.2.15	142.250.183.78	TLSv1.2	95 Application Data
63	18.116586433	142.250.183.78	10.0.2.15	TCP	62 443 → 41082 [ACK] Seq=1 Ack=40 Win=65535
64	18.127811965	142.250.183.78	10.0.2.15	TLSv1.2	95 Application Data
65	18.175024883	10.0.2.15	142.250.183.78	TCP	56 41082 → 443 [ACK] Seq=40 Ack=40 Win=62780
66	19.118083902	10.0.2.15	142.251.42.66	TLSv1.2	95 Application Data
67	19.118386474	142.251.42.66	10.0.2.15	TCP	62 443 → 37266 [ACK] Seq=1 Ack=40 Win=65535
68	19.129121029	142.251.42.66	10.0.2.15	TLSv1.2	95 Application Data
69	19.172596646	10.0.2.15	142.251.42.66	TCP	56 37266 → 443 [ACK] Seq=40 Ack=40 Win=62780
70	20.480844042	10.0.2.15	172.217.174.227	TCP	56 [TCP Dup ACK 1#2] 59722 → 80 [ACK] Seq=1
71	20.409397251	172.217.174.227	10.0.2.15	TCP	62 [TCP Dup ACK 2#2] [TCP ACKed unseen segme
72	20.980842492	10.0.2.15	172.217.174.227	TCP	56 [TCP Dup ACK 3#2] 56174 → 80 [ACK] Seq=1
73	20.900893744	172.217.174.227	10.0.2.15	TCP	62 [TCP Dup ACK 4#2] [TCP ACKed unseen segme
74	22.011906312	10.0.2.15	172.217.174.227	TCP	56 [TCP Dup ACK 5#2] 56192 → 80 [ACK] Seq=1
75	22.011930161	10.0.2.15	172.217.174.227	TCP	56 [TCP Dup ACK 6#2] 56204 → 80 [ACK] Seq=1
76	22.012147007	172.217.174.227	10.0.2.15	TCP	62 [TCP Dup ACK 7#2] [TCP ACKed unseen segme

Frame 56: 95 bytes on wire (760 bits), 95 bytes captured (760 bits) on interface any, id 0

TLS v1.v3: it is designed to prevent eavesdropping, tampering, and message forgery

10490	162.616708422	10.0.2.15	192.124.249.24	TCP	56 [TCP Keep-Alive] 3
10491	162.617103768	104.18.21.226	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10492	162.617103917	192.124.249.24	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10493	162.801134947	10.0.2.15	34.120.24.7	TLSv1.3	85 Application Data
10494	162.801646095	34.120.24.7	10.0.2.15	TCP	62 443 → 42288 [ACK]
10495	163.081083867	34.120.24.7	10.0.2.15	TLSv1.3	81 Application Data
10496	163.081105052	10.0.2.15	34.120.24.7	TCP	56 42288 → 443 [ACK]
10497	163.129538761	10.0.2.15	35.213.12.39	TCP	56 [TCP Keep-Alive] 3
10498	163.130090271	35.213.12.39	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10499	163.641285031	10.0.2.15	104.83.196.216	TCP	56 [TCP Keep-Alive] 4
10500	163.641775587	104.83.196.216	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10501	163.890887003	10.0.2.15	142.250.183.67	TCP	56 [TCP Keep-Alive] 5
10502	163.900516442	142.250.183.67	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10503	165.177009512	10.0.2.15	172.64.155.188	TCP	56 [TCP Keep-Alive] 4
10504	165.177099347	172.64.155.188	10.0.2.15	TCP	62 [TCP Keep-Alive AC
10505	165.045062321	10.0.2.15	54.182.1.73	TCP	56 [TCP Keep-Alive] 5
10506	165.045467916	54.182.1.73	10.0.2.15	TCP	62 [TCP Keep-Alive AC

Frame 10495: 81 bytes on wire (648 bits), 81 bytes captured (648 bits) on interface any, id 0

Linux cooked capture v1

Internet Protocol Version 4, Src: 34.120.24.7, Dst: 10.0.2.15

Transmission Control Protocol, Src Port: 443, Dst Port: 42288, Seq: 5797, Ack: 1461, Len: 25

Transport Layer Security

Estimating the RTT:

No.	Time	Source	Destination	Protocol	Length	Info
35	3.085315257	172.217.167.164	10.0.2.15	TLSv1.3	87	Application Data
36	3.085322053	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1565 Ack=862 Win=63910 Len=0
37	3.086272357	10.0.2.15	172.217.167.164	TLSv1.3	87	Application Data
38	3.086443103	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=862 Ack=1596 Win=65535 Len=0
39	3.179557612	172.217.167.164	10.0.2.15	TLSv1.3	2165	Application Data, Application Data
40	3.179598572	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=2971 Win=62780 Len=0
41	3.181092698	172.217.167.164	10.0.2.15	TLSv1.3	87	Application Data
42	3.181119208	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=3002 Win=62780 Len=0
43	3.182180658	172.217.167.164	10.0.2.15	TLSv1.3	95	Application Data
44	3.182199896	10.0.2.15	172.217.167.164	TCP	56	44438 → 443 [ACK] Seq=1596 Ack=3041 Win=62780 Len=0
45	3.184943938	10.0.2.15	172.217.167.164	TLSv1.3	95	Application Data
46	3.185429658	172.217.167.164	10.0.2.15	TCP	62	443 → 44438 [ACK] Seq=3041 Ack=1635 Win=65535 Len=0
47	24.097913389	fe80::b4e4:c843:92a	ff02::2	ICMPv6	64	Router Solicitation
48	28.002286595	10.0.2.15	35.244.181.201	TLSv1.2	102	Application Data

We will choose 41 and 42th packet

We can clearly see that our local host (10.0.2.15) started a TCP connection to the server (172.217.167.164) and then it send the message as TLSv1.3

RTT = time when 43rd packet started moving – time when 41st packet started moving

$$= 3.182199896 - 3.181892698$$

$$= 0.0003072 \text{ ms}$$

This tells us that it is very fast